

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
4 May 2006 (04.05.2006)

PCT

(10) International Publication Number
WO 2006/047133 A2

(51) International Patent Classification:
G06F 17/30 (2006.01)

(21) International Application Number:
PCT/US2005/037351

(22) International Filing Date: 17 October 2005 (17.10.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/621,178 22 October 2004 (22.10.2004) US
10/988,014 12 November 2004 (12.11.2004) US

(71) Applicant (for all designated States except US): **STREAM THEORY, INC.** [US/US]; 19600 Fairchild Road, Suite 350, Irvine, CA 92612 (US).

(72) Inventor: **DE VRIES, Jeffrey**; 902 West Olive Avenue, Sunnyvale, CA 94086 (US).

(74) Agent: **AHMANN, William, F.**; Perkins Coie LLP, 101 Jefferson Drive, Menlo Park, CA 94025 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SYSTEM AND METHOD FOR PREDICTIVE STREAMING

(57) Abstract: A technique for predictive streaming involves receiving a request for a block associated with a streaming application and serving data associated with the block. A block request database is checked to predict what block is likely to be requested next based upon prior block request data. The predicted block may be identified when serving the data associated with the requested block. A system developed according to the technique includes a streaming server, a block request database, and a prediction engine that uses the block request database to predict block requests. The streaming server provides data associated with the predicted block request.

WO 2006/047133 A2

SYSTEM AND METHOD FOR PREDICTIVE STREAMING

BACKGROUND

Software streaming involves downloading small pieces of files as the pieces are needed by the program being streamed. These small pieces may be referred to as blocks. A streaming client sends requests for blocks as they are needed up to a streaming server, which sends back streaming data that is associated with the requested block. Sending a request and receiving the streaming data may cause delays that can slow down the streamed program.

There are many problems associated with streaming software that it would be advantageous to negate, work around, or reduce. For example, predicting blocks for streaming has not been satisfactorily addressed.

SUMMARY

A technique for predictive streaming involves receiving a request for a first block of a streaming application, checking a block request database, predicting a second block request based on the block request database, and sending, in response to the request, data associated with the first block and data associated with the second block. A block may be an arbitrarily large portion of a streaming application. The block request database includes probabilities or means for determining probabilities that the second block will be requested given that the first block has been requested. One or more factors may be considered when determining the probability of a request for the second block. Data sent in response to the request may include data associated with the first block and data associated with the second block. The data associated with the first block and the data associated with the second block may not be analogous. In an embodiment, the data associated with the first block is responsive to a block request for the first block, while the data associated with the second block is data sufficient to facilitate making a request for the second block.

In an embodiment, the technique may further include predicting block requests based on the block request database, then sending data associated with the block requests. In another embodiment, the data associated with the second block includes data sufficient to render a request for the second block unnecessary. In another embodiment, the technique further includes piggybacking the data associated with the second block on a reply to the request for the first

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 depicts a networked system for use in an exemplary embodiment;

FIG. 2 depicts a computer system for use in the system of FIG. 1;

FIG. 3 depicts a portion of the computer system of FIG. 2 and components of the system of FIG. 1;

FIGS. 4 and 5 depict flowcharts of exemplary methods for predictive streaming according to embodiments;

FIG. 6 depicts a conceptual view of a system 600 for providing streaming data and identifying a predicted block in response to a block request according to an embodiment;

FIGS. 7A to 7C and 8A to 8B depict exemplary request logs and block request databases according to embodiments;

FIGS. 9 to 12 depict exemplary methods according to alternative embodiments;

FIG. 13 depicts an exemplary log request and block request database according to an embodiment.

components providing the web server functionality and the server functionality provided by the server computer system 106, which will be described further below.

Access to the network 102 is typically provided by Internet service providers (ISPs), such as the ISPs 110 and 116. Users on client systems, such as client computer systems 112, 118, 122, and 126 obtain access to the Internet through the ISPs 110 and 116. Access to the Internet allows users of the client computer systems to exchange information, receive and send e-mails, and view documents, such as documents which have been prepared in the HTML format. These documents are often provided by web servers, such as web server 104, which are referred to as being "on" the Internet. Often these web servers are provided by the ISPs, such as ISP 110, although a computer system can be set up and connected to the Internet without that system also being an ISP.

Client computer systems 112, 118, 122, and 126 can each, with the appropriate web browsing software, view HTML pages provided by the web server 104. The ISP 110 provides Internet connectivity to the client computer system 112 through the modem interface 114, which can be considered part of the client computer system 112. The client computer system can be a personal computer system, a network computer, a web TV system, or other computer system. While Fig. 1 shows the modem interface 114 generically as a "modem," the interface can be an analog modem, isdn modem, cable modem, satellite transmission interface (e.g. "direct PC"), or other interface for coupling a computer system to other computer systems.

Similar to the ISP 114, the ISP 116 provides Internet connectivity for client systems 118, 122, and 126, although as shown in Fig. 1, the connections are not the same for these three computer systems. Client computer system 118 is coupled through a modem interface 120 while client computer systems 122 and 126 are part of a LAN 130.

Client computer systems 122 and 126 are coupled to the LAN 130 through network interfaces 124 and 128, which can be ethernet network or other network interfaces. The LAN 130 is also coupled to a gateway computer system 132 which can provide firewall and other Internet-related services for the local area network. This gateway computer system 132 is coupled to the ISP 116 to provide Internet connectivity to the client computer systems 122 and 126. The gateway computer system 132 can be a conventional server computer system.

Alternatively, a server computer system 134 can be directly coupled to the LAN 130 through a network interface 136 to provide files 138 and other services to the clients 122 and 126, without the need to connect to the Internet through the gateway system 132.

The computer system 140 is one example of many possible computer systems which have different architectures. For example, personal computers based on an Intel microprocessor often have multiple buses, one of which can be an I/O bus for the peripherals and one that directly connects the processor 148 and the memory 152 (often referred to as a memory bus). The buses are connected together through bridge components that perform any necessary translation due to differing bus protocols.

Network computers are another type of computer system that can be used with the present invention. Network computers do not usually include a hard disk or other mass storage, and the executable programs are loaded from a network connection into the memory 152 for execution by the processor 148. A Web TV system, which is known in the art, is also considered to be a computer system according to the present invention, but it may lack some of the features shown in FIG. 2, such as certain input or output devices. A typical computer system will usually include at least a processor, memory, and a bus coupling the memory to the processor.

In addition, the computer system 140 is controlled by operating system software which includes a file management system, such as a disk operating system, which is part of the operating system software. One example of an operating system software with its associated file management system software is the family of operating systems known as Windows® from Microsoft Corporation of Redmond, Washington, and their associated file management systems. Another example of operating system software with its associated file management system software is the Linux operating system and its associated file management system. The file management system is typically stored in the non-volatile storage 156 and causes the processor 148 to execute the various acts required by the operating system to input and output data and to store data in memory, including storing files on the non-volatile storage 156.

Some portions of the detailed description are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise

include the network 162. In another alternative, the computer system 140 may be physically or wirelessly coupled to the streaming client 164.

The streaming client 164 may be coupled to and accessible through the network 162. The streaming client 164 could be a software, firmware, or hardware module. Alternatively, the streaming client 164 could include some combination of software, firmware, or hardware components. The streaming client 164 may be part of a computer system, such as the computer system 140 (FIG. 2). The streaming client 164 may include a processor, a memory, and a bus that couples the processor to the memory. The streaming client 164, or a computer system associated with the streaming client 164, may make use of certain programs when executing a streaming application. For example, a streaming application may be intended for use with a specific version of DirectX™, Acrobat™, or QuickTime™, which typically is installed prior to executing the streaming application.

The computer system 140 includes a processor 166, a memory 168, and a bus 170 that couples the processor 166 to the memory 168. The memory 168 may include both volatile memory, such as DRAM or SRAM, and non-volatile memory, such as magnetic or optical storage. The memory 168 may also include, for example, environment variables. The processor 166 executes code in the memory 168. The memory 168 includes a streaming server 172, a block request database 174, a prediction engine 176, and one or more streaming applications 178. Some or all of the programs, files, or data of the computer system 140 could be served as Web content, such as by the server computer 106 (FIG. 1). The programs, files, or data could be part of a server computer on a LAN or WAN, such as the server computer 134 (FIG. 1).

The streaming server 172 may be configured to serve data associated with a block of a streaming application, such as one of the streaming applications 178, in response to a request for the block. The block request database 174 may include data associated with block requests received from one or more streaming clients, such as the streaming client 164. The block request database 174 may include a block request log, which is updated with each or with a subset of block requests. The block request database 174 may be client-specific or generally used by all or a subset of all streaming clients from which block requests are received. The block request database 174 may be application-specific or generally associated with all or a subset of all of the streaming applications 178. The block request database 174 may include a block request history that has been derived from data associated with block requests received over time, from some original default values, or from data input by, for example, a user or software agent that

want or need. For example, if the streaming server 172 predicts that a streaming client 164 will need streaming data associated with a first block, and sends streaming data associated with the first block, the streaming client 164 is unable to choose whether to receive the streaming data based on, for example, local factors. On the other hand, if the streaming client 164 receives identifying data, the streaming client 164 may determine whether it actually wants the streaming data. In an alternative embodiment, the streaming server 172 may sometimes serve streaming data associated with a predicted block right away, possibly without even receiving a request from the streaming client 164 for the predicted block.

The prediction engine 176 may give a weight to a number of blocks. For example, the prediction engine 176 may predict a first block that is more likely to be requested than a second block (a probability parameter). The streaming client 164 may first request the blocks with, for example, the highest probability of being requested. As another example, the prediction engine 176 may give greater weight to a first block over a second block if the prediction engine 176 predicts the first block will be requested sooner than the second block (a temporal parameter). As another example, the prediction engine 176 may give greater weight to a first block over a second block if the first block is larger than the second block (a size parameter). The prediction engine 176 may weigh parameters associated with the streaming client 164, as well. For example, if the streaming client 164 has a limited buffer size, the aggressiveness parameter may be set lower. As another example, if the download bandwidth is low, the prediction engine 176 may place more weight on temporal or size parameters. Alternatively, the streaming client 164 may manage block requests according to local conditions, while the prediction engine 176 acts the same for all or a subset of all streaming clients.

In operation, the streaming client 164 sends over the network 162 to the streaming server 172 a request for a block associated with a streaming application of the streaming applications 178. The streaming server 172, or some other component (not shown) of the computer system 140, logs the request. For the purposes of example only, the block request database 174 is treated as a block request log. However, in various embodiments, the block request database 174 could be derived from a block request log or from individual or groups of block requests that are not recorded in a log. In an exemplary embodiment, the block request database 174 includes parameters derived from previous block requests, or from default or initial settings, that aid in predicting subsequent block requests from the streaming client 164.

at module 194 with requesting the second block. It is assumed for the purposes of example that local factors merit the request for the second block. For example, if streaming data associated with the second block is in a local disk cache, the second block may not be requested. The flowchart continues at module 196 with receiving the request for the second block. The flowchart continues at module 198 with second a reply that includes block data associated with the second block. The flowchart ends at module 200 with receiving the second block data. It should be noted that, though the flowchart may be thought of as depicting sequential small reads, as opposed to depicting building large blocks, the blocks could very well be large. This method and other methods are depicted as serially arranged modules. However, modules of the methods may be reordered, or arranged for parallel execution as appropriate.

When a streaming server sends data associated with a predicted block (second block) request along with data associated with a requested block (first block), the first and second block data may be thought of as a "large block" if they are queued for sending back-to-back. However, in order to queue the first and second blocks back-to-back, the streaming server would have to receive the request for the second block and queue the second block data before the streaming server was finished sending the first block data. When data is sent as a large block, the streaming server can have nearly continuous output. Large blocks can help to more fully utilize available bandwidth, since large data requests can fully "saturate the pipe." This is in contrast to sequential requests for blocks, which may not saturate the pipe because of the pause between sending first block streaming data and receiving a request for and sending second block streaming data. Fully saturating the pipe can improve performance.

In some cases, a block may be made large initially. For example, if a streaming application is for a level-based game, a large block may include data associated with an entire level. When a request for the block is received, data associated with the entire level is returned as continuous output from the streaming server. Alternatively, large blocks can be built "on the fly" based on a first block request and predicted block requests. For example, if the streaming server sends identifying data for multiple predicted block requests piggybacked on a reply to a first block request, along with streaming data associated with the first block, the streaming client can consecutively request some or all of the multiple predicted blocks. If the requests are made in relatively rapid succession, the streaming server may queue streaming data associated with two predicted blocks more rapidly than the streaming server sends streaming data associated

of time are not referred to as continuous. The flowchart ends at module 218 with receiving the streaming data. The streaming client may receive the streaming data. Though the streaming server saturates the pipe, the streaming client may or may not receive a continuous stream of data, depending on various factors that are well-understood in the art of data transmission, such as network delays, and are, therefore, not described herein.

At times, a streaming client may rapidly request blocks in a strictly sequential manner. For example, if a sequence of blocks is associated with a video clip, the blocks are reasonably likely to be served, in order, one after the other. A streaming server may recognize a sequential pattern of block requests, either because the streaming server is provided with the pattern, or because the streaming server notices the pattern from block requests it receives over time. The streaming server may send the pattern to the streaming client in response to a block request that has been found to normally precede block requests for blocks that are identified in the pattern. Using the pattern, the streaming client may predictively request additional blocks in anticipation of the streaming application needing them. The streaming client may make these requests in relatively rapid succession, since each request can be made using the pattern the streaming client received from the streaming server. This may result in output pipe saturation at the streaming server. The streaming client may or may not wait for a reply to each request. Predictively requested blocks may be stored in a local cache until they are needed. The parameters that control recognition of the pattern, as well as how aggressive the read-ahead schedule should be, can be independently specified at the file, file extension, directory, and application level. In addition, it can be specified that some blocks are predictively downloaded as soon as a file is opened (even before seeing a read) and that the open call itself should wait until the initial blocks have been downloaded. In an exemplary embodiment, the pattern includes identifying data for each block represented in the pattern. The patterns may be included in a block request database.

In an alternative embodiment, the streaming server may provide the streaming client with one or more patterns as a pattern database. The streaming server may or may not provide the pattern database when the streaming client first requests streaming of a streaming application associated with the pattern database. The streaming client may use the pattern database to predict which blocks to request based on block requests it intends to make. For example, if a first block request is associated with blocks in a pattern in the pattern database, the streaming client requests the first block and each of the blocks in the pattern in succession. The pattern database may be included in a block request database.

included in the system 600. The multiple streaming applications could be discretely managed (e.g., by allowing only predictions of blocks from the same streaming application as a requested block) or as a collection of blocks (e.g., a predicted block could be from a streaming application that is different from that of the requested block). The streaming applications could be local or remote. The output node 230 may be an interface for connecting the system 600 to other computer systems, or a logical node within the system 600 for sending block requests from the streaming server 220. The output node 230 may or may not be treated as part of the streaming server 220. The output node 230 and the input node 218 may or may not be treated as components of an input/output node.

In operation, the input node 219 receives a block request from a streaming client (not shown). The input node 219 provides the block request to the streaming server 220. The streaming server 220 logs the block request in the request log 222. The streaming server obtains a prediction from the prediction engine 224. The logging of the request and the obtaining of a prediction need not occur in any particular order. For example, the streaming server 220 could obtain a prediction from the prediction engine 224 prior to (or without consideration of) the logged block request.

The prediction engine 224 checks the request log 222, performs calculations to represent data in the request log 222 parametrically, and updates the parameters of the block request database 226. The prediction engine 224 checks the parameters of the block request database 226 in order to make a prediction as to subsequent block requests from the streaming client. The checking of the request log and checking of the parameters need not occur in any particular order. For example, the prediction engine 224 could check the parameters prior to checking the request log and updating the parameters. The prediction engine 224 could check the request log and update the parameters as part of a routine updating procedure, when instructed to update parameters by a user or agent of the system 600, or in response to some other stimulus, such as an access to the streaming application 228. Accordingly, the checking of the log request and updating of the parameters may be thought of as a separate, and only indirectly related, procedure vis-à-vis the checking of parameters to make a prediction. The prediction may be in the form of one or more block IDs, identifying data for one or more blocks, a pattern, or any other data that can be used by a streaming client to determine what blocks should be requested predictively.

certain probability. For example, an aggressiveness parameter may be set to a cut-off threshold of, e.g., 0.5. If predictive certainty for a block does not meet or exceed the cut-off threshold, the prediction engine 224 may not provide the prediction for the block to the streaming server 220.

FIGS. 7A to 7C are intended to help illustrate how to predict a block request based on the requested block. FIGS. 7A to 7C depict exemplary block request logs and parameters derived therefrom according to embodiments. FIGS. 7A to 7C are not intended to illustrate preferred embodiments because there are many different items of data that could be recorded or omitted in the request log and many different parameters that could be derived from selected items of data.

FIG. 7A depicts an exemplary request log 232 and a block request database 234. In this example, the request log 232 includes a listing of blocks in the order the blocks were requested from a streaming client. For exemplary purposes, the request log 232 is assumed to be associated with a single streaming client. The request log 232 may or may not be associated with a single streaming application. The block request database 234 includes parameters derived from the request log 232. For exemplary purposes, the parameters are derived only from entries in the request log 232. In alternative embodiments, the parameters could include default values or otherwise rely on data that is not included in the request log 232. For the purposes of example, the request log 232 is assumed to include only the values shown in FIG. 7A. For the purposes of example, a block request (the current block request) is considered for each of the blocks 3 to 8.

Current block request for block 3: As illustrated in the request log 232, a block request for block 5 (the second block request) immediately follows the request for block 3 (the first block request). Since the second block request follows the request for block 3, a prediction can be made about whether the current block request (for block 3) will be followed by a request for block 5. Since, for the purposes of example, the parameters are derived only from the request log 232 (and no other data is considered), it might be assumed that a request for block 5 is 100% certain. The predicted block parameters array for block 3 is [(5, 1.0)]. This can be interpreted to mean, following a request for block 3, the probability of a request for block 5 is 1.0. Of course, this is based on a small data sample and is, therefore, subject to a very large error. However, over time the probability may become more accurate.

Current block request for block 4: As illustrated in the request log 232, a block request for block 4 has not been made. Accordingly, no prediction can be made.

FIG. 7B depicts an exemplary request log 236 and a block request database 238 according to another embodiment. In this example, the request log 236 includes a listing of blocks in the order the blocks were requested from a streaming client, plus a time field that represents what time the blocks were sent by a streaming client. In an alternative, the time field may represent what time the blocks were received, logged, or otherwise managed. The block request database 238 includes a predicted block parameters array with a time field, which represents the time difference between when a first block request and a second block request were sent. For example, for the predicted block parameters array for block 3, the time field is 6:18:04, which is intended to mean 6 hours, 18 minutes, and 4 seconds. This is the difference between when block 3 and block 5 were sent, as shown in the request log 236. As another example, the predicted block parameters array for block 8 is [(3, 0.5, 0:05:20), (6, 0.5, 5:10:15)]. This can be interpreted to mean that blocks 3 and 6 are equally likely to be requested following a request for block 8. However, since there is a time entry, weight can be given to the entry with the lower associated time difference. Since, with respect to a streaming application, 5 hours is a long time, weight may be given to block 3 (requested about 5 minutes after block 8) over block 6 (requested about 5 hours after block 8). Indeed, in this example, since the time between block 6 and block 8 is so long, a prediction engine could choose to assume that there really isn't a predictive relationship between blocks 6 and 8.

In an embodiment, time-related predictive parameters may be ignored if they are too high. The threshold value over which a time difference would result in a block request being ignored may be referred to as a temporal aggressiveness parameter. For example, if a temporal aggressiveness parameter is 1 hour, then the predicted block parameters array for block 8 could be rewritten as [(3, 1.0, 0:05:20)]. That is, the values related to block 6 are ignored since block 6 was received more than 1 hour after block 8, according to the request log 236. Alternatively, the predicted block parameters array for block 8 could be rewritten as [(3, 0.5, 0:05:20)], which is basically the same, but the probability is not recalculated when the values related to block 6 are ignored. Similarly, the predicted block parameters array for block 3 could be rewritten as [], since there are no block requests within 1 hour (the temporal aggressiveness threshold, in this example) of the block request for block 3.

In an alternative embodiment, a prediction engine could keep chaining predicted blocks back through the prediction engine to get subsequent predicted blocks, under the assumption that the "first round" predicted blocks were correct. The probabilities of subsequent predicted blocks

Each block associated with a streaming application has a probability of being requested when streaming the streaming application that is based on how many times the block is requested when streaming the streaming application, compared to the total number of times the streaming application has been streamed. For the purposes of example, one instance, from beginning to end, of streaming a streaming application may be referred to as a session. In a given session, one or more blocks, in one or more combinations, may be requested.

A session may be thought of in terms of the block requests made over the course of streaming a streaming application. For example, in the five sessions depicted in the log request 244 of FIG. 8A, Session 1 consists of three block requests: 7, 8, and 3; Session 2 consists of three block requests: 8, 3, and 4; Session 3 consists of three block requests: 5, 8, and 6; Session 4 consists of two block requests: 8 and 3; and Session 5 consists of two block requests: 8 and 3. Considering only these five sessions, a probability associated with a block request may be determined by adding the number of times a block is requested in a session and dividing the sum by the total number of sessions (in this case, five). For example, as shown in the block request database 246, Block 3, which is included in Sessions 1, 2, 4, and 5, has a zero order probability of 0.8. Blocks 4, 5, 6, and 7 each have a zero order probability of 0.2 because they are included only in Session 2, 3, 3, and 1, respectively. Since Block 8 is included in each of the Sessions 1-5, Block 8 has a zero order probability of 1.0. In an embodiment, probabilities are constructed over multiple sessions by multiple users to obtain composite probabilities.

In some embodiments, it may be desirable to utilize a higher order probability. For example, if the streaming application is a game program that starts in a room with four doors that lead to four different rooms, each of which is associated with multiple block requests, the multiple blocks associated with each of the four different rooms may be equally likely (about 25% each). If a door is selected, each of the multiple blocks associated with each of the doors (even those not taken) may be about 25%. By using a first-order predictor, once we see the first block request for one of the rooms, the subsequent blocks for that room will have first-order probabilities near 100%, and so the multiple blocks for that room can be predictively downloaded.

FIG. 8B is intended to illustrate first order probabilities. FIG. 8B depicts a request log 248 and a block request database 250. When determining first order probabilities, a first block request is used as context. In other words, if a first block request is made, the probability of a second block request may be calculated by comparing the probability, based upon previous

It should be noted that in certain embodiments, it may not be possible to incorporate all incoming block requests into the request log because block requests are being "filtered" by the local disk cache. This may result in the request log including only those block requests for blocks that weren't in the local disk cache, which will throw the predictors off. Accordingly, in an embodiment, a streaming server may indicate to a streaming client that the server is interested in collecting prediction data. In this case, the streaming client would send a separate data stream with complete block request statistics to the streaming server. In this example, the separate data stream may include all actual block requests made by the application, regardless of whether that request was successfully predicted or is stored in a local cache of the streaming client.

FIG. 9 depicts a flowchart of an exemplary method for updating predictive parameters. The flowchart starts at module 252 with informing a streaming client of an interest in collecting prediction data. This request may or may not be included in a token file. The request may be sent to a streaming client when the streaming client requests streaming of an application. After module 252, the flowchart continues along two paths (254-1 and 254-2). The modules 254 may occur simultaneously, or one may occur before the other.

The flowchart continues at module 254-1 with receiving block requests. The streaming server may receive the block requests from the streaming client. The flowchart continues at module 256-1 with sending data associated with the block requests, including predictions, if any. The flowchart continues at decision point 258-1, where it is determined whether the session is over. If the session is not over, the flowchart continues from module 254-1 for another block request. Otherwise, if the session is over, the flowchart ends for modules 254-1 to 258-1.

The flowchart continues at module 254-2 with receiving block request statistics, including block requests for blocks stored in a local disk cache. Module 254-2 may or may not begin after module 258-1 ends. Module 254-2 may or may not continue after module 258-1 ends. The flowchart continues at module 256-2 with updating predictive parameters using the block request statistics. The predictive parameters may be used at module 256-1 to provide predictions. The flowchart continues at decision point 258-2, where it is determined whether the session is over. If the session is not over, the flowchart continues from module 254-2 for more block request statistics. Otherwise, if the session is over, the flowchart ends for modules 254-2 to 258-2.

In another embodiment, a streaming client may indicate to a streaming server that the streaming client is interested in receiving predictive block data IDs. Then the streaming server

exemplary method for receiving predictive data from a streaming client. The flowchart begins at module 278 with receiving block requests. The block requests may be from a streaming client. The flowchart continues at module 280 with sending block requests, including predictions, if any, to the streaming client. The flowchart continues at decision point 282, where it is determined whether the session is over. If the session is not over (282-N), then the flowchart continues at module 278, as described previously. If, on the other hand, the session is over (282-Y), then the flowchart continues at module 284 with receiving predictive data from the streaming client. The predictive data may include a request log, a block request history, or one or more parameters derived from block request data. The flowchart ends at module 286 with updating a block request database using the predictive data. In this way, the streaming server can adapt the block request database in response to each session.

In an alternative embodiment, a streaming server can maintain a block request database that keeps track of "runs." Runs are sets of blocks for which the block requests occur closely spaced in time. For example, if a sequence of blocks is requested within, say, 0.5 second of a preceding block request in the sequence, the sequence of blocks may be referred to as a run. Runs can be used to efficiently utilize memory resources by recording probabilities on the run level instead of per block. Runs can also be used to reduce the amount of predictive download data. For example, for a level-based game, a user may download a first block, then the rest of the game will be predictively downloaded for each level, since the probabilities of downloading each level are, for the purposes of this example, nearly 100%.

By keeping track of runs, after the first level has been downloaded, predictive downloads can be "shut off" until the start of blocks for the second level begin. For example, if the blocks at the beginning of the run, which trigger or signal the run, are detected, the block IDs for the rest of the run may be sent to the streaming client, who then chain requests them. However, those subsequent block requests do not trigger any further run. So, the predictive downloads cause the blocks in the middle of a run to not act as a triggering prefix of another run; no further predictions are made from the middle of a run.

FIG. 13 is intended to illustrate maintaining a block request database that includes runs. FIG. 13 depicts a request log 288 and a block request database 290. Parameters associated with blocks and runs are omitted from the block request database 290 so as to more clearly focus on the point being illustrated. The omitted parameters could be any parameters derived from the request log for the purpose of facilitating the prediction of future block requests (e.g., first-order

CLAIMS

What is claimed is:

1. A method, comprising:
receiving a request for a first block of a streaming application;
checking a block request database;
predicting a second block request based on the block request database;
sending, in response to the request, data associated with the first block and data associated with the second block.
2. The method of claim 1, further comprising predicting a plurality of block requests based on the block request database, wherein said sending further includes sending data associated with the plurality of block requests.
3. The method of claim 1, wherein the data associated with the second block is sufficient to identify the second block so as to facilitate making a request for the second block.
4. The method of claim 1, wherein the data associated with the second block includes data sufficient to render a request for the second block unnecessary.
5. The method of claim 1, further comprising piggybacking the data associated with the second block on a reply to the request for the first block.
6. The method of claim 1, further comprising:
logging the request for the first block; and
updating the block request database to incorporate data associated with the logged request.
7. The method of claim 6, wherein the logged request is a first logged request, further comprising setting a temporal aggressiveness parameter, wherein the block request database is updated to incorporate data associated with the first logged request to the extent that associations with a second logged request is made if a difference between a receive time associated with the

block based upon a first block request for a first block and predictive parameters associated with the first block;

a streaming server, coupled to the prediction engine, that is configured to:

obtain the prediction about the second block request from the prediction engine,

include data associated with the second block in a response to the first block request, in addition to data associated with the first block, and

send the response in reply to the first block request.

16. The system of claim 15, wherein the data associated with the second block is sufficient to identify the second block so as to facilitate making the second block request.

17. The system of claim 15, wherein the data associated with the second block includes data sufficient to render the second block request unnecessary.

18. The system of claim 15, wherein the streaming server is further configured to piggyback the data associated with the second block on a reply to the first block request.

19. The system of claim 15, further comprising a request log, wherein the streaming server is further configured to log the first block request in the request log.

20. The system of claim 19, wherein the prediction engine is further configured to update the block request database according to the request log.

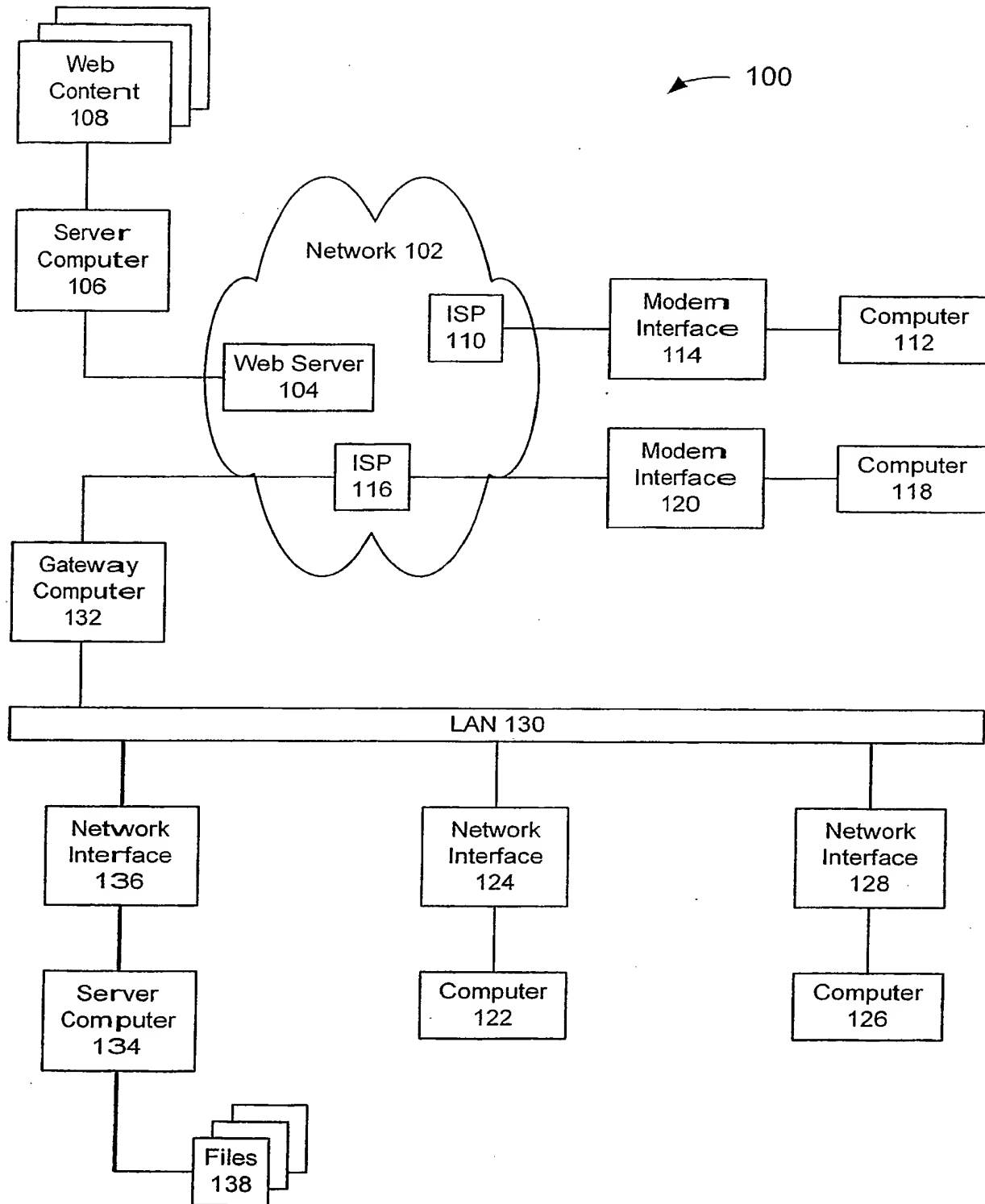


FIG. 1

140 →

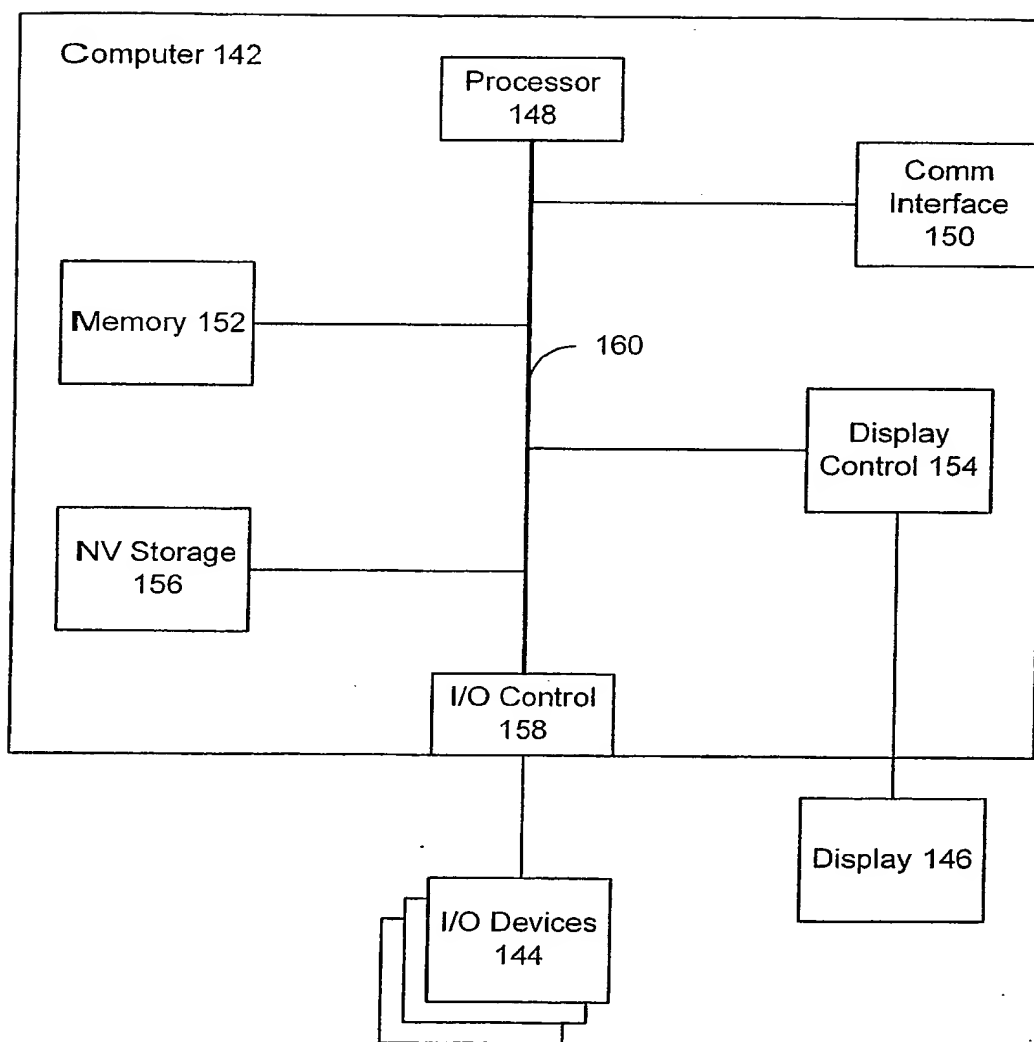


FIG. 2

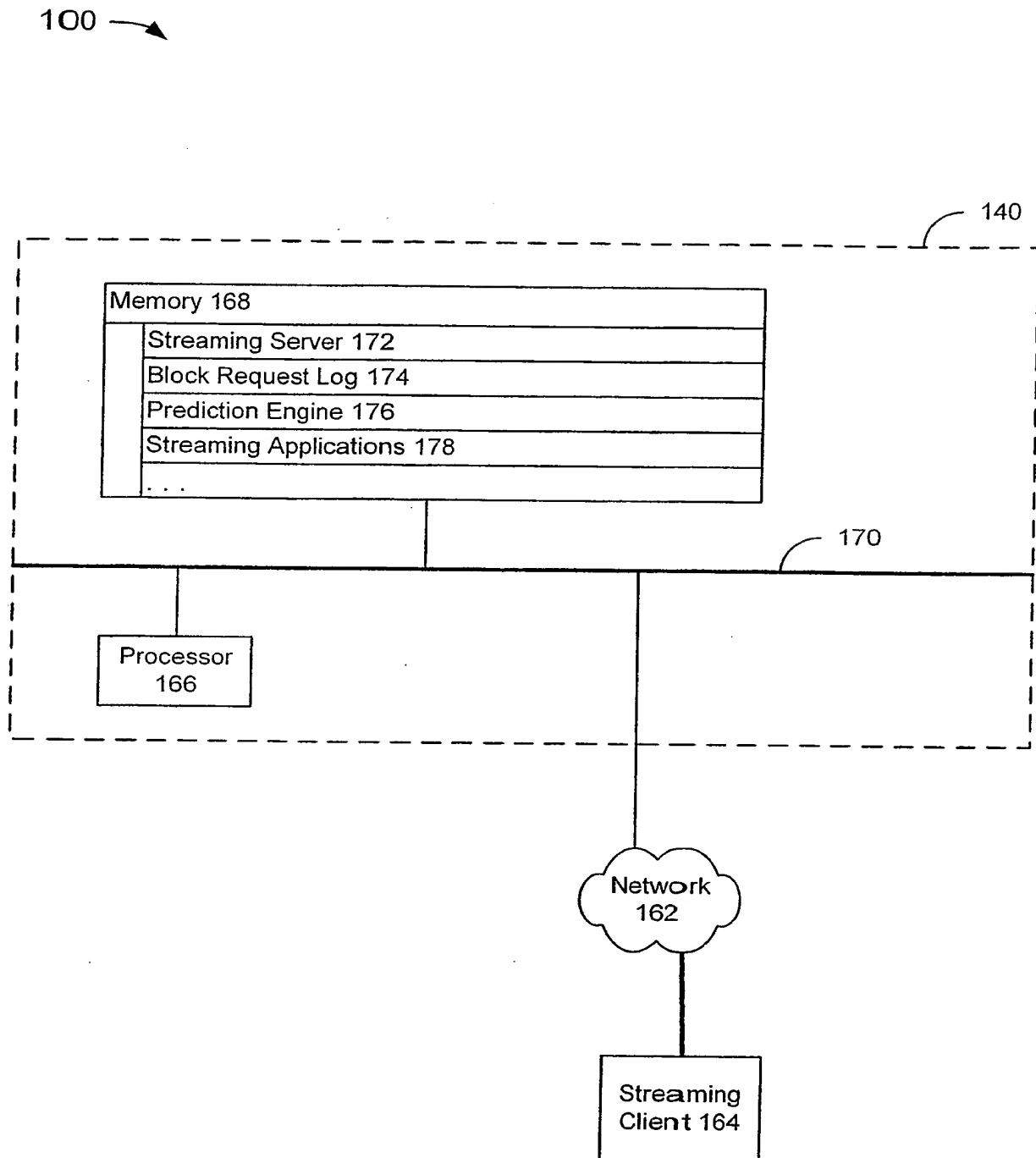


FIG. 3

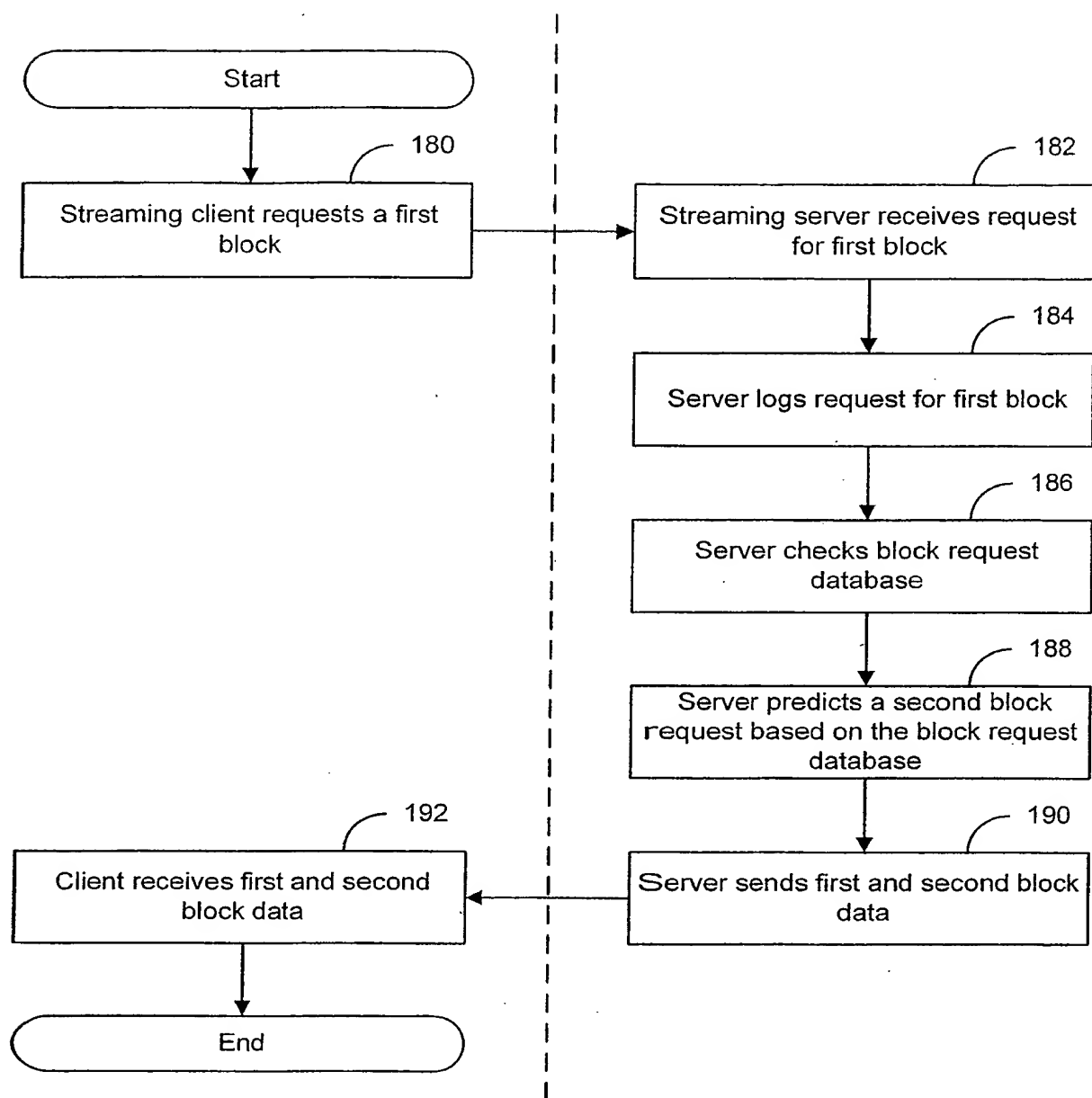


FIG. 4

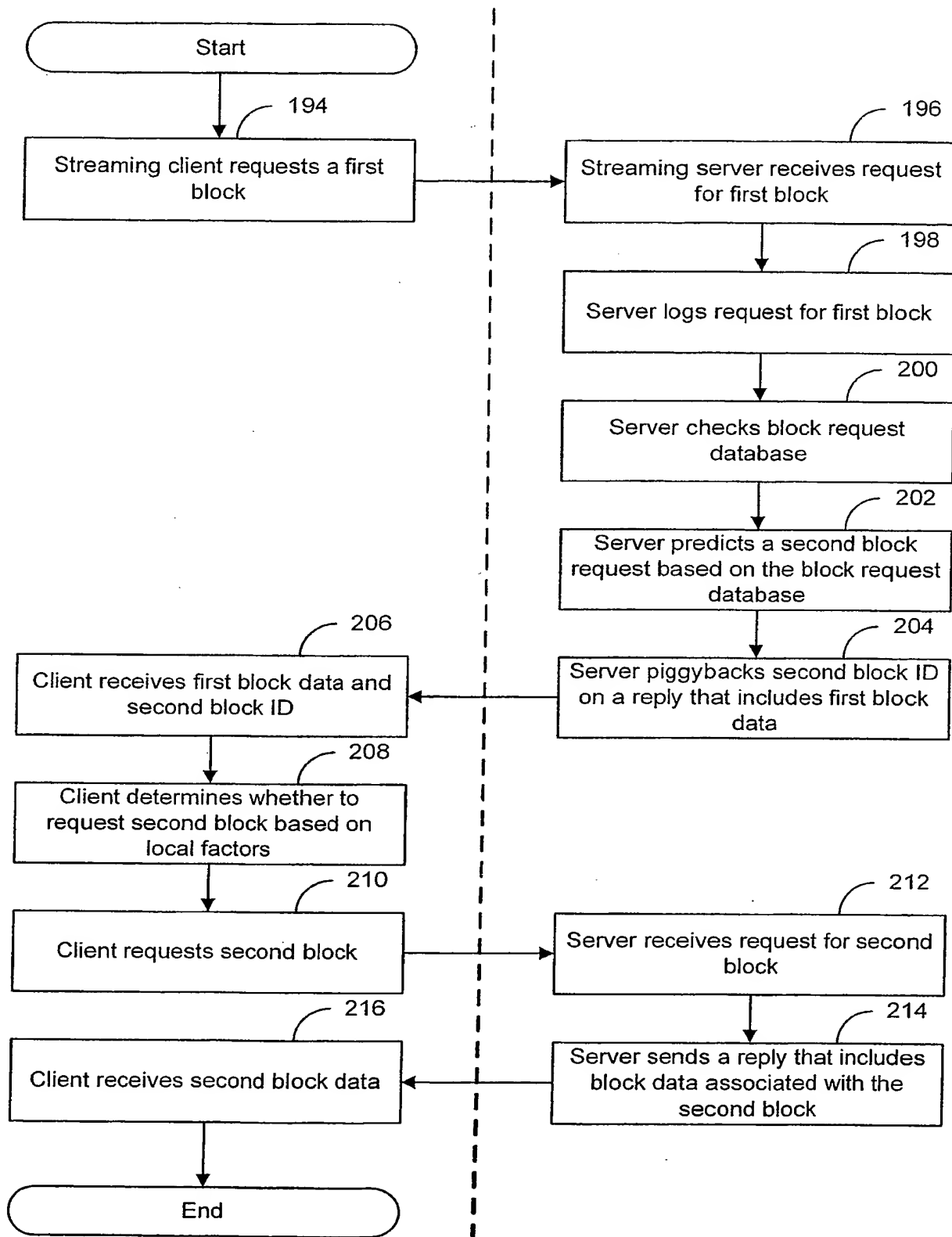


FIG. 5

600 →

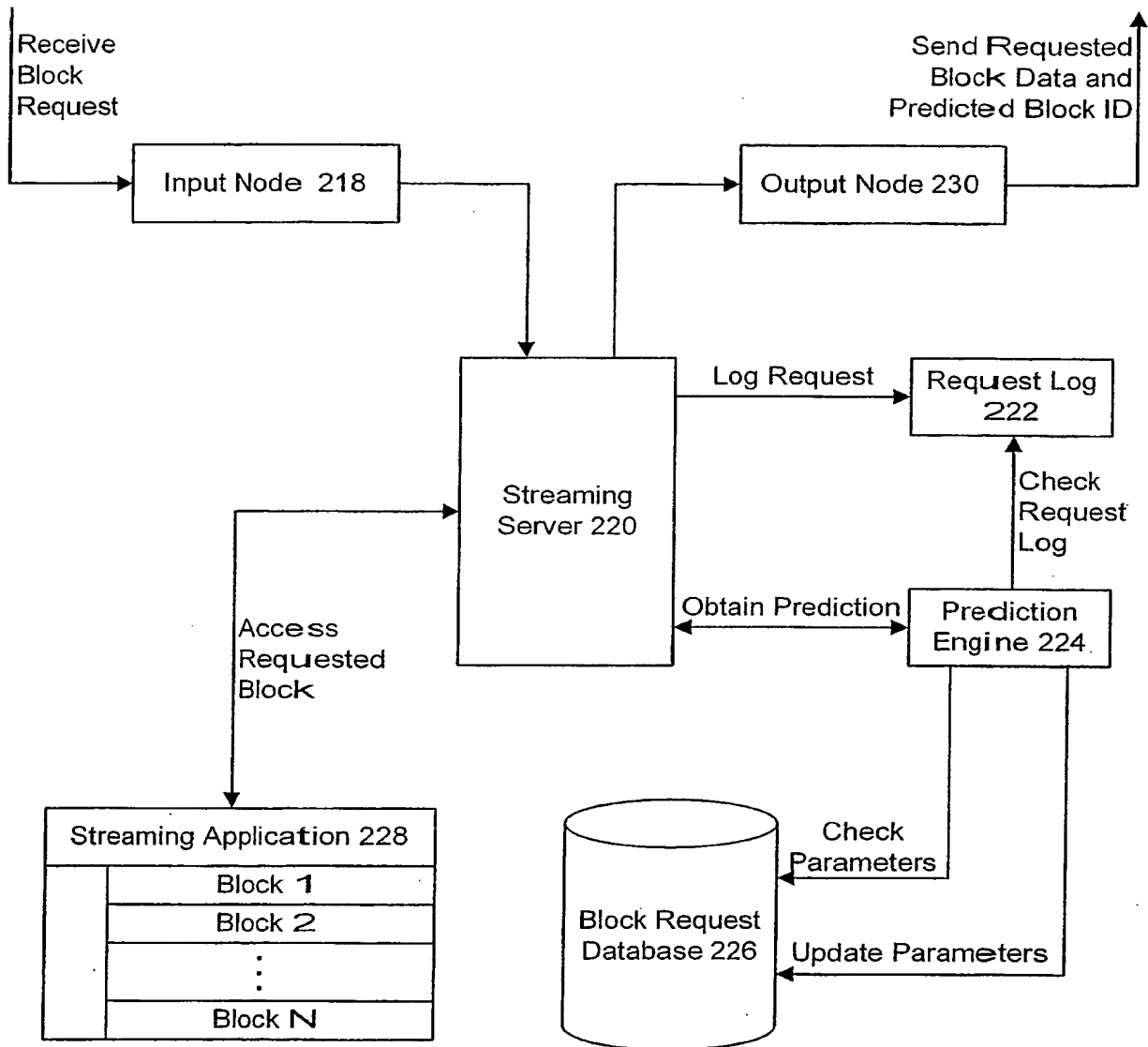


FIG. 6

232

Request Log	
Block	
7	
8	
3	
5	
8	
6	

234

Block Request Database	
Block	Predicted Block Parameters Array
3	[(5, 1.0)]
4	[]
5	[(8, 1.0)]
6	[]
7	[(8, 1.0)]
8	[(3, 0.5), (6, 0.5)]

FIG. 7A

236

Request Log	
Block	Time
7	01:03:12
8	01:05:36
3	01:10:56
5	07:29:00
8	07:32:30
6	11:42:45

238

Block Request Database	
Block	Predicted Block Parameters Array
3	[(5, 1.0, 6:18:04)]
4	[]
5	[(8, 1.0, 0:03:30)]
6	[]
7	[(8, 1.0, 0:02:24)]
8	[(3, 0.5, 0:05:20), (6, 0.5, 5:10:15)]

FIG. 7B

240

Request Log		
Client	Block	Time
1	7	01:03:12
1	8	01:05:36
2	8	01:10:25
1	3	01:10:56
2	3	01:11:18
2	4	01:18:23
1	5	07:29:00
1	8	07:32:30
1	6	11:42:45
3	8	12:05:36
2	8	12:10:25
3	3	12:10:56
2	3	12:11:47

242

Block Request Database	
Block	Predicted Block Parameters Array
3	[(4, 0.5, 0:07:05), (5, 0.5, 6:18:04)]
4	[(8, 1.0, 10:52:02)]
5	[(8, 1.0, 0:03:30)]
6	[]
7	[(8, 1.0, 0:02:24)]
8	[(3, 0.8, 0:00:53), (6, 0.2, 5:10:15)]

FIG. 7C

244

Request Log	
Block	
7	} Session 1
8	
3	
8	} Session 2
3	
4	
5	} Session 3
8	
6	
8	} Session 4
3	
8	} Session 5
3	

246

Block Request Database	
Block	Zero-Order Probability
3	0.8
4	0.2
5	0.2
6	0.2
7	0.2
8	1.0

FIG. 8A

248

Request Log	
Block	
7	} Session 1
8	
3	
8	} Session 2
3	
4	
5	} Session 3
8	
6	
8	} Session 4
3	
8	} Session 5
3	

250

Block Request Database						
Block	First-Order Probability					
	3	4	5	6	7	8
3	n/a	1.0	0	0	1.0	0.8
4	0.25	n/a	0	0	0	0.2
5	0	0	n/a	1.0	0	0.2
6	0	0	1.0	n/a	0	0.2
7	0.25	0	0	0	n/a	0.2
8	1.0	1.0	1.0	1.0	1.0	n/a

FIG. 8B

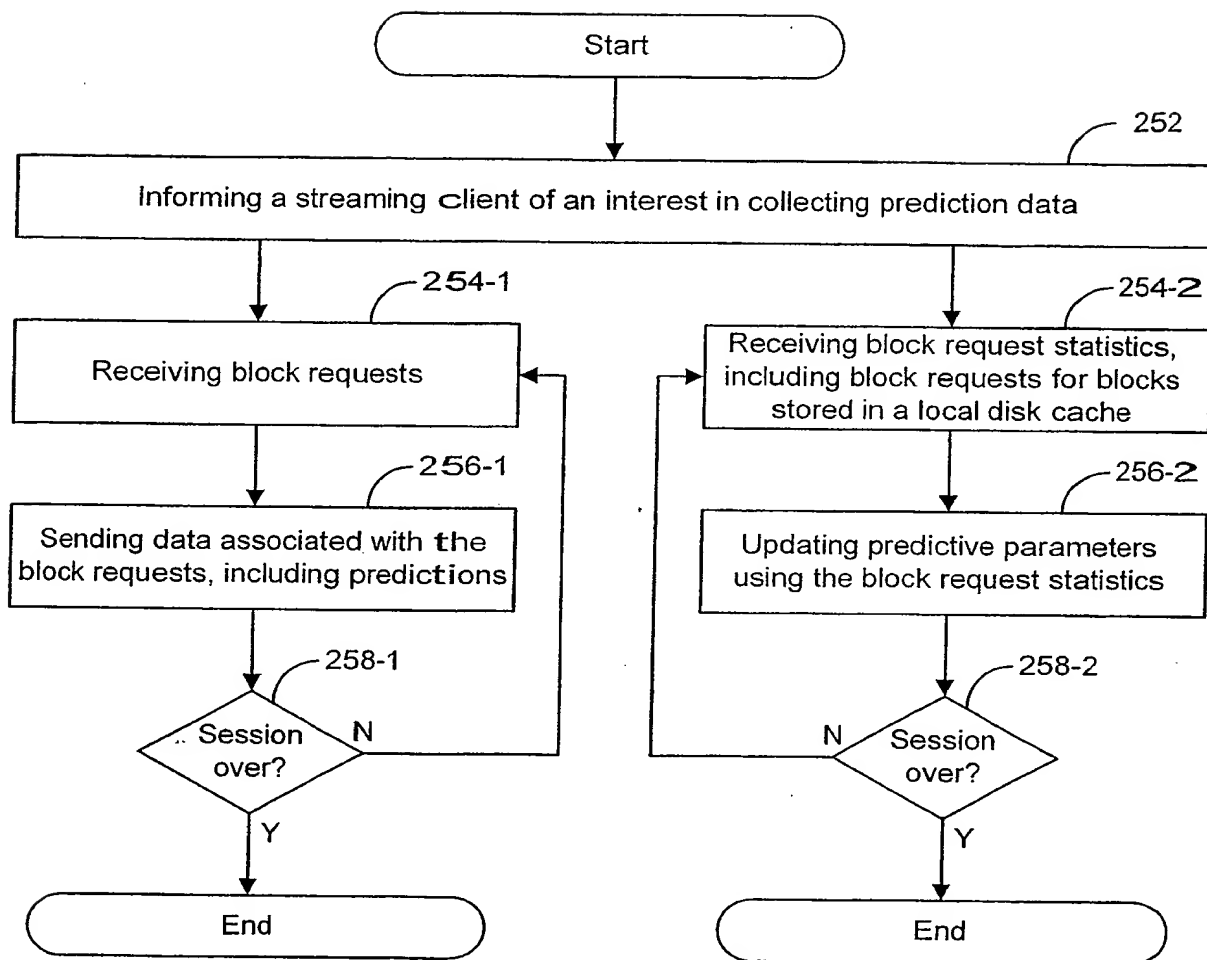


FIG. 9

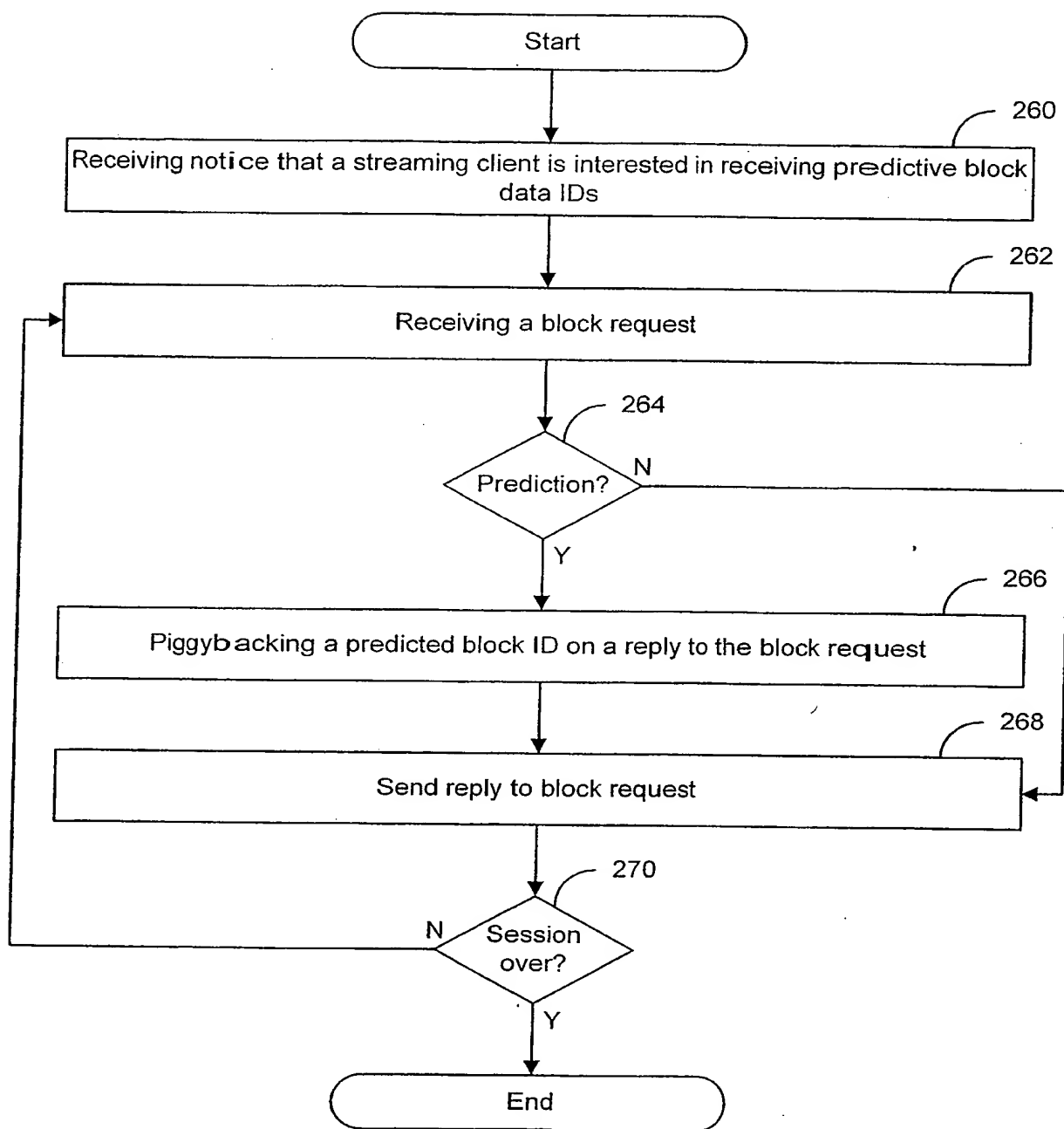


FIG. 10

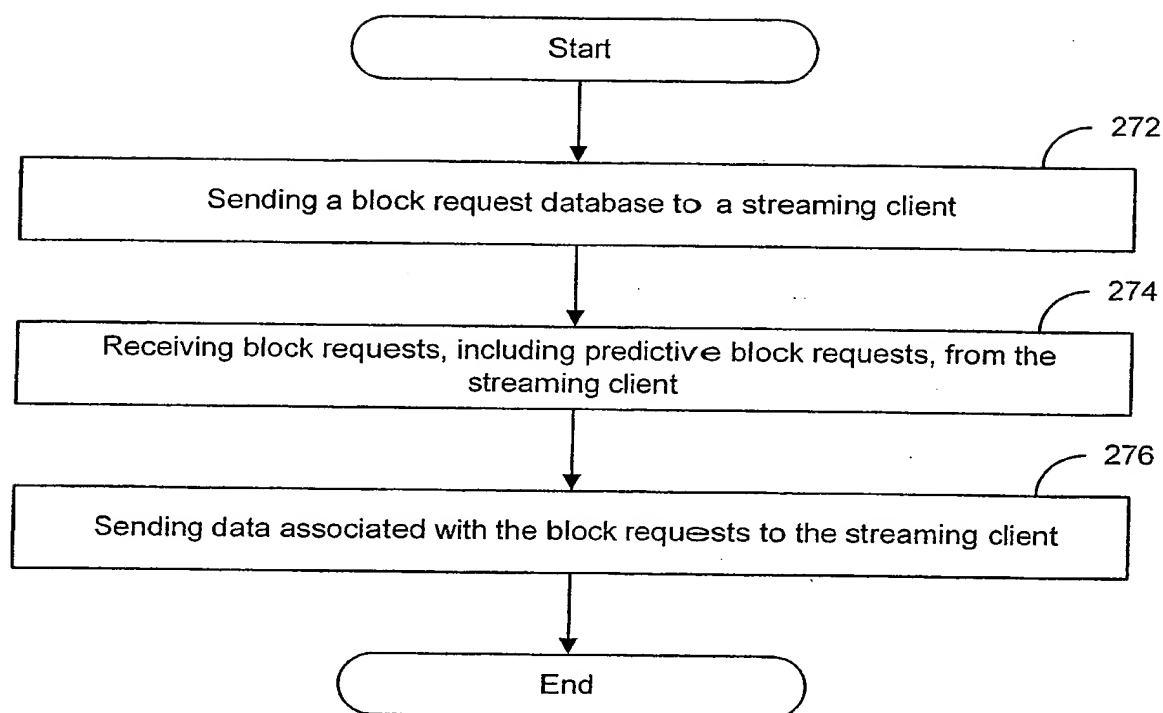


FIG. 11

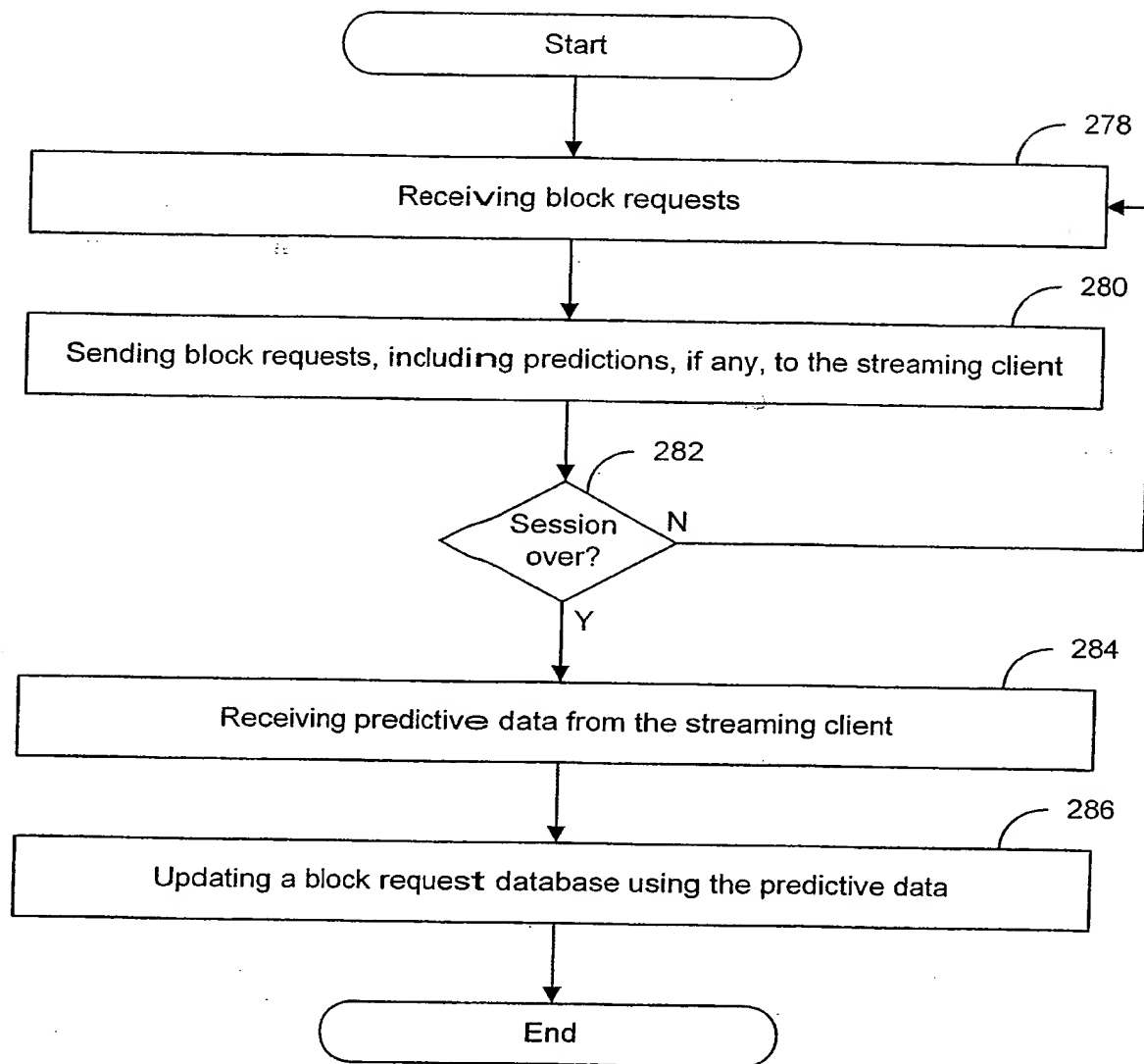


FIG. 12

Request Log		
Session	Block	Time
1	16	2:59:52
1	17	3:00:01
1	18	3:00:02
1	19	3:00:03
1	20	3:00:04
1	21	3:00:05
1	22	3:00:06
1	23	3:00:07
1	24	3:00:08
1	25	3:00:09
1	15	3:00:12
2	31	4:59:52
2	17	5:00:01
2	18	5:00:02
2	19	5:00:03
2	26	5:00:04
2	27	5:00:05
2	28	5:00:06
2	29	5:00:07
2	30	5:00:08
2	32	5:00:17

Block Request Database	
Block	Parameters
15	
16	
17-25	

Block Request Database	
Block	Parameters
17-19, 26-30	
31	
32	

Block Request Database	
Block	Parameters
15	
16	
17-19	
20-25	
26-30	
31	
32	

FIG. 13